EXHIBIT 2-1

android developers

Designing for Performance

In this document

Introduction
Optimize Judiciously
Avoid Creating Unnecessary Objects
Performance Myths
Prefer Static Over Virtual
Avoid Internal Getters/Setters
Use Static Final For Constants
Use Enhanced For Loop Syntax
Consider Package Instead of Private Access with Inner Classes
Use Floating-Point Judiciously
Know And Use The Libraries
Use Native Methods Judiciously
Closing Notes

An Android application will run on a mobile device with limited computing power and storage, and constrained battery life. Because of this, it should be *efficient*. Battery life is one reason you might want to optimize your app even if it already seems to run "fast enough". Battery life is important to users, and Android's battery usage breakdown means users will know if your app is responsible draining their battery.

Note that although this document primarily covers micro-optimizations, these will almost never make or break your software. Choosing the right algorithms and data structures should always be your priority, but is outside the scope of this document.

Introduction

There are two basic rules for writing efficient code:

- Don't do work that you don't need to do.
- Don't allocate memory if you can avoid it.

Optimize Judiciously

This document is about Android-specific micro-optimization, so it assumes that you've already used profiling to work out exactly what code needs to be optimized, and that you already have a way to measure the effect (good or bad) of any changes you make. You only have so much engineering time to invest, so it's important to know you're spending it wisely.

(See Closing Notes for more on profiling and writing effective benchmarks.)

This document also assumes that you made the best decisions about data structures and algorithms, and that you've also considered the future performance consequences of your API decisions. Using the right data structures and algorithms will make more difference than any of the advice here, and considering the performance consequences of your API decisions will make it easier to switch to better implementations later (this is more important for library code than for application code).

(If you need that kind of advice, see Josh Bloch's *Effective Java*, item 47.)

One of the trickiest problems you'll face when micro-optimizing an Android app is that your app is pretty much guaranteed to be running on multiple hardware platforms. Different versions of the VM running on different processors running at different speeds. It's not even generally the case that you can simply say "device X is a factor F faster/slower than device Y", and scale your results from one device to others. In particular, measurement on the emulator tells you very little about performance on any device. There are also huge differences between devices with and without a JIT: the "best" code for a device with a JIT is not always the best code for a device without.

If you want to know how your app performs on a given device, you need to test on that device.